APPENDIX H

EXPRESS MAIL LABEL
NO.: EL386270487US

Appendix H will be understood and appreciated from the following detailed description, taken in conjuction with the drawings in which:

Fig. 70 is a simplified illustration of a chain of blocks structure in a windowing reference management tool, constructed and operative in accordance with a preferred embodiment of the present invention; and

Fig. 71 is a simplified illustration of the indexing main structure of a windowing reference management tool, constructed and operative in accordance with a preferred embodiment of the present invention.

1. Introduction

I.I Scope

This tool for Windowing Reference Management aims to support efficient storage and retrieval operations of window data. The goal of this tool is to allow the storage of simple windowing data items which are inserted in order into an internal data structure, in such a way that allows their efficient retrieval.

Retrieving data efficiently is a common goal of many systems, especially those that call for a massive data processing and manipulation.

The motivation for the development of this specific tool arises from various application requirements arising from the inspection of BGAs and lead frames.

1.2 Glossary And Acronyms

• Template - an automated way to generate customized classes and functions.

• STL - Standard Template Library - a template-based library of generic C++ data structures and methods.

• Container - a class supporting data storage. Examples of STL containers include:

• vectors

• lists « queues and priority queues>>

• A container adapter class - classes that encapsulate an existing container, while providing a new user interface.

• A. function object - (template) object class with an operator () defined.

2. Concepts

A possible approach to the problem of efficient retrieval of windowing data from reference, in accordance with a preferred embodiment of the present invention is derived naturally from the observation that scanned image data is usually ordered; first,

by lines (y coordinate value) , then by the order within each line (the x coordinate value). Assuming data items have a key value according to which the items can be sorted, or ordered, the task of extracting information from windowing reference can be reduced to the task of extracting information from an ordered reference, as described herein. The basic prerequisites typically include:

• Data items have key values according to which items can be sorted (ordered). The key can be the value of the data item itself, in case of numeric data types. Keys are of a scalar data type, e.g., int, double, char, accessible by a function object. This requirement aims to provide a uniform access to the key, independently of data type definition. Thus, the order by which data items are ordered is subject to the key function supplied by the user.

• Data is given in order, that is, items are inserted in order into the local container either one by one, or an entire ordered container is inserted.

Thus the development of a reference tool arising from a preferred application needs to provide an efficient retrieval of data from containers that do not have an efficient method for data access: Various containers may not allow for an efficient data retrieval. One example of such container is the Chain (if Blocks container, a vector like container, for which memory is allocated in blocks, each containing a fixed number of elements). Blocks are allocated in chunks, each of which contains a fixed number of blocks (the Chain of Blocks structure is very similar to that of a linked list, see Figure 70). A direct (random) access to items in the chain is not possible. An adapter that can wrap the Chain of Blocks, while allowing it to retain its insertion functionality (e.g., push_back), with the benefit of an efficient retrieval of items is clearly used.

Next a design may be considered for a tool for indexing of ordered data; The main goal of such a tool is to support efficient retrieval of ordered data from reference. The reference can be external, e.g. a file, or internal — reference data located in memory. Moreover, the use of such a tool may be directly, or indirectly —via a container adapter, wrapping the container while allowing efficient access to its data items. To allow for efficient access of data, the indexing system is based on a number of indices, defined on construction. Each index corresponds to another sub range of possible key values, or bin. The bounds of the range of key values is defined by the minimum and maximum of possible key values over the items in the container to be indexed. The range is then sub

divided into equal size bins, whose number is specified by the used number of indices. Each index directs towards the data item whose key is the first to reside in the corresponding sub range.

2.1 Interfaces

The indexing tool is an STL/C++ independed library, therefore it can only be used as part of a C++ program.

2.2 Data

The following is a description of the integral data members involved in the indexing tool:

• unsigned int n_bins; The number of indices (bins) used for the indexing system.

• Bins Index', The indexing system: A vector of n_bins indices, or bins, covering the used range of possible key values, as defined by the following parameters:

• pair<Key::resulf_type, Key::result_iype> Xrange', The range over which the indexing system is defined, where Key is an object function allowing to access the data-item key. This range is defined by the minimum and maximum of possible key values over the items in the container to be indexed.

• Bins size. For each bin, the number of data-item keys that reside in that bin is stored. Note that only the index of the first data item whose key resides in that sub range need to be stored at each bin because data is ordered, thus the Bins size does not really reflect storage usage, but a potential one, i.e., it indicates the distribution of data items over the entire indexing system.

2.3 Error Handling

Errors are handled by return codes. Possible errors and their return codes are given in Table 1.

Table 1: Errors and Warnings Codes and Meanings

Function	Code	Error/Warning
Create index	Err(l)	Container data items are not ordered
load store	Err(2)	xternal device is not readable External device is not writeable

Create index	Err(3)	Empty Container - Nothing to index

3. Design Description

The indexing of ordered data tool is a template class; Two versions of this class exist: (a) A Memory Indexing object class whose indices direct into a Container in memory, and (b) A File Indexing object class whose indices direct into a file containing the data items to be indexed. Note that in the latter case, the indices are actually offsets in file. The File Indexing version is templated with a Key function object, whereas the Memory Indexing class is also templated with the Container. The following are the methods provided by both classes:

Construction

• Purpose - Construction and initialization.

• Inputs - A resolution parameter which defines the number of indices (or bins) to be used by the internal indexing system.

• Processing - The internal indexing system is initialized with the specified number of indices (bins).

• Outputs - None.

Create

• Purpose - Create the internal system

• Inputs - None.

• Processing - Define the range to cover using the minimum and maximum over key values over the items residing in the local container, calculate the range covered by each bin, and assign the value of the indices accordingly. For each bin, the corresponding index stores a reference to the first data item whose key is inside this specific bin range. The order of the data items is checked to verify that they are given in order. Note: the key values can only be accessed via the Key function object, with which the Indexing class is templated.

• Outputs - None.

Update

• Purpose - Update the internal system.

• Inputs - The new number of bins to use for resolution over that same range.

• Processing - Update the number of bins (indices) used by the internal indexing system, and re-create the indices accordingly.

• Outputs - None.

Query

• Purpose - Efficient retrieval of information out of the local container.

• Inputs - The query range.

• Processing - Using the indices, calculate the first and last bins in which data items satisfying the query range may reside.

• Outputs - A pair of bin numbers, between which data items are expected to satisfy the query criterion.

Storage Operations Serialization

• Purpose - Store the indexing system information.

• Inputs - The external device into which to direct the serialization and storage operation.

• Processing - Since the indices of the File indexing object are offsets, storing them is straightforward. For the Indexing in Memory case, serialization is a prerequisite. Consequently, the Memindexing has also the serialize and deserialize methods, converting its indices into offsets that can be stored into and loaded from and external device.

• Outputs - None.

Deserialization

• Purpose - Deserialize and load indexing information from external device.

• Inputs - The external device specification.

• Processing - Deserialize data, and load the indexing system information from the specified external device and restore it into memory.

• Outputs - None.